Engineering of Advanced Software Solutions (EASS)

HIT, Israel

Yossi Eliaz

2022

- Admin
- Technical debt
- Business logic
- Bash and commandline (based on MIT's missing semester)
- Git, GitHub
- Interactive class

Admin stuff

- Discord
- Github account
- HW on Github
- Creating a Canvas account (https://canvas.instructure.com/)
- Accepted invitation from AWS Academy and GitHub (I have sent links)
- AWS on Cavnvas
- LinkedIn
- Commandline (WSL)
- Docker
- Moodle (minimal interaction over there)
- Volunteer to summarize the lectures
- Stackoverflow
- Engagment on Discord
- Hackernews

References:

Missing Semester MIT
AWS cloud certificate

What is Technical Debt?

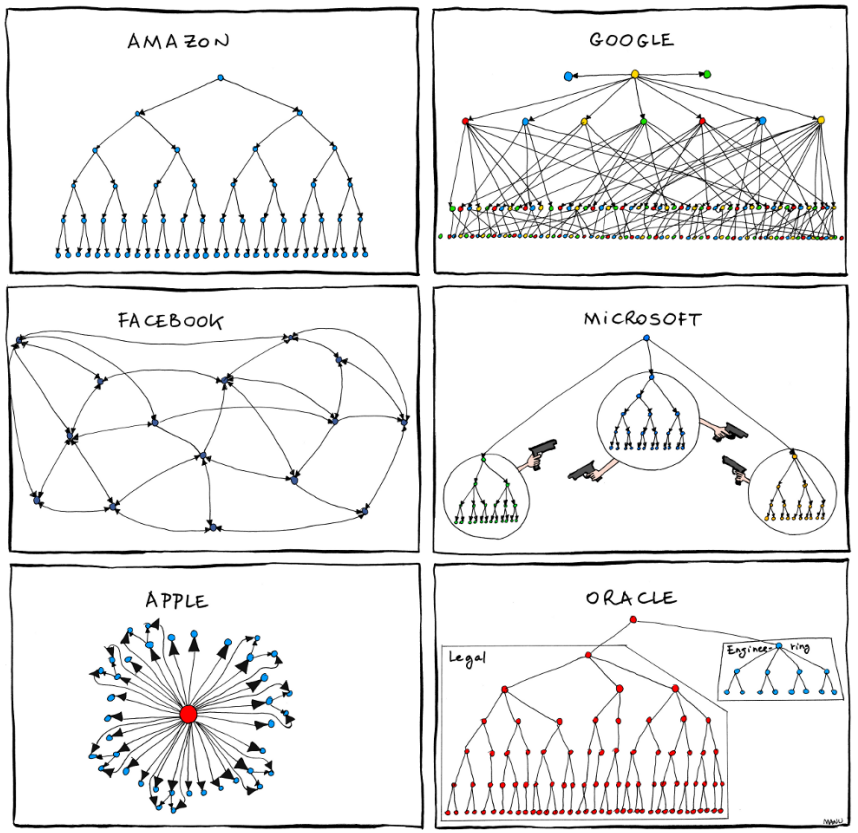
- “In software development, there is always a constant need to balance speed and quality. Some quality will always have to be sacrificed to release features within a reasonable timeframe, so any of these shortcuts will often be tasked as future projects. Those unattended tasks become what is called technical debt.”
- “There are several reasons why technical debt happens. Product owners may focus more on the need to implement and release new features and less on fixing past problems or create a generic enough infrastructure to support future developments. In some severe cases, product owners completely underestimate the outcomes of dealing with poor infrastructure, bugs and poorly designed software.”
- “Ultimately, technical debt can sometimes lead to software users having bad experiences and thereby increasing user churn rates. Together, a lack of developer awareness and task ownership can lead to more technical debt.”

References:

https://logz.io/blog/technical-debt/

Business logic

- “**Business rules** are what your non-software developers tell you what your software needs to do.”
- “**Business logic** is the part of your code that specifically implements business rules.”



References:

https://softwareengineering.stackexchange.com/questions/234251/what-really-is-the-business-logic
http://www.ritholtz.com

Intro to commandline and tools (interactive)

- Vim
- Bash
- Git
- Docker

Important tools and commands

- `echo`, `while`, `find`, `vars`, `printenv`, `htop`, shebang, wild cards
- `cp`, `touch`, `mkdir`, `ls`, `uniq`, `awk`, `rm`
- `man man`
- `brew`
- `wget`
- `curl`

References:

https://missing.csail.mit.edu/2020/shell-tools/

Sneak peek to docker

- `docker run`
- `docker ps`
- `docker run -ti`

Stackoverflow good questions usually have

Must have:

1. Problem statement
2. Sample code and data
3. Spelling, grammar and formatting

Example:

https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array

References:

https://codeblog.jonskeet.uk/2010/08/29/writing-the-perfect-question/
https://stackoverflow.com/help/how-to-ask

First task

- Checkout github classroom and the first task about git and github
  https://classroom.github.com/classrooms/99552739-eass-hit-2022-part-a

AWS course (due a week after Passover == April 30, 2022)

- S3, EC2, RDS, and EBS modules
- must get 100 on all 4 modules
- grading will be 25% per module

A bit more about EC2 instances and types of hardwares (HW)

EC2 provides secure, resizable compute cloud services. It makes web-scale cloud computing easier and offers HW such as:

- ARM vs. Intel vs. AMD (x86, x86_64)
- GPUs (Nvidia, Intel)
- TPUs (on Google Computing Platform)
- Metal instacnes on AWS
- FPGA-based nodes

Instance Types and prices (useful links)

https://aws.amazon.com/ec2/instance-types/
https://instances.vantage.sh/

Instance Types (summary)

1. General Purpose
2. Compute Optimised
3. Memory Optimised
4. Accelerated Computing (P instances are for general-purpose GPU applications)

Pricing

There are four ways to pay for EC2 instances: On-Demand, Reserved Instances, and Spot Instances & Per-Second Billing. You can also pay for Dedicated Hosts which provide you with EC2 instance capacity on physical servers dedicated for your use.

First task on Git and GitHub

Checkout github classroom and the first task about git and github

https://classroom.github.com/classrooms/99552739-eass-hit-2022-part-a

How to test our code/system

General approaches for testing

- Static vs. Dynamic
- Passive testing
- White-box vs. Black-box testing

Types of testing coverage metric

- API testing – testing all public and private APIs Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)

Types of tesing systems (CI/CD)

- Unit vs. Integration testing
- System testing
- Compatibility testing
- Installation testing
- Smoke and sanity testing
- Regression testing

We will use pytest and fastapi testing system

- https://fastapi.tiangolo.com/tutorial/testing/
- https://docs.pytest.org/

All exercises

1. 4 modules on AWS course (S3, EC2, EBS, RDS) - if you finish all the course you get +10 bonus points to final grade
2. Build full REST/HTTP fastapi backend + Dockerization (due 1/4)
3. UI (react/streamlit) (due 1/5)
4. Docker compose the server with UI and backend plus server and write a clear README with git submodules (due 29/5)
5. Presentation of the system in a demo in a 2-3 minutes video on youtube and clear README (due 29/5)

Ideas for porjects next semester (based on skills we will learn this semester)

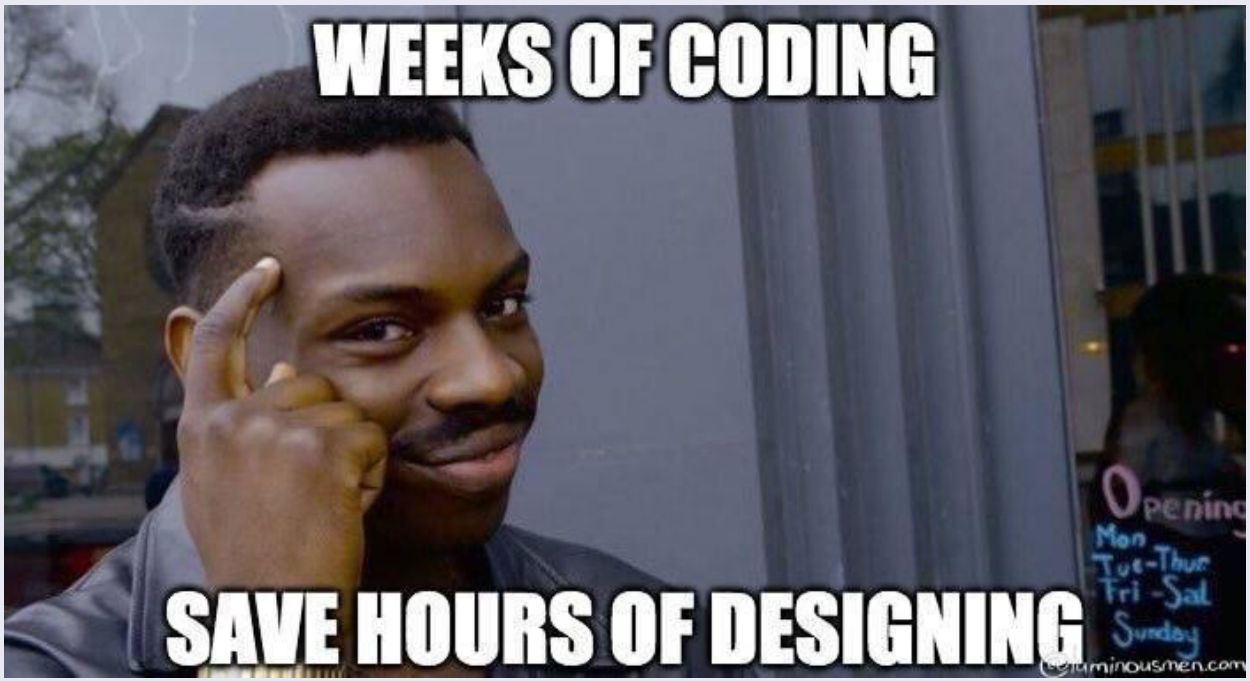
1. AI/ML based predictive system
2. Smart contracts (web3)
3. Any other system with at least 3 microservices

List of the subjects in our course

- Monolithic vs. Microservices
- Docker
- Client-Server
- REST/HTTP API
- FastAPI
- Pytest
- asyncio
- Frontend (React javascript and Streamlit python)
- Docker compose
- Functional programming
- How to compile a new library

What Are The Best Software Engineering Principles?

Measure twice and cut once

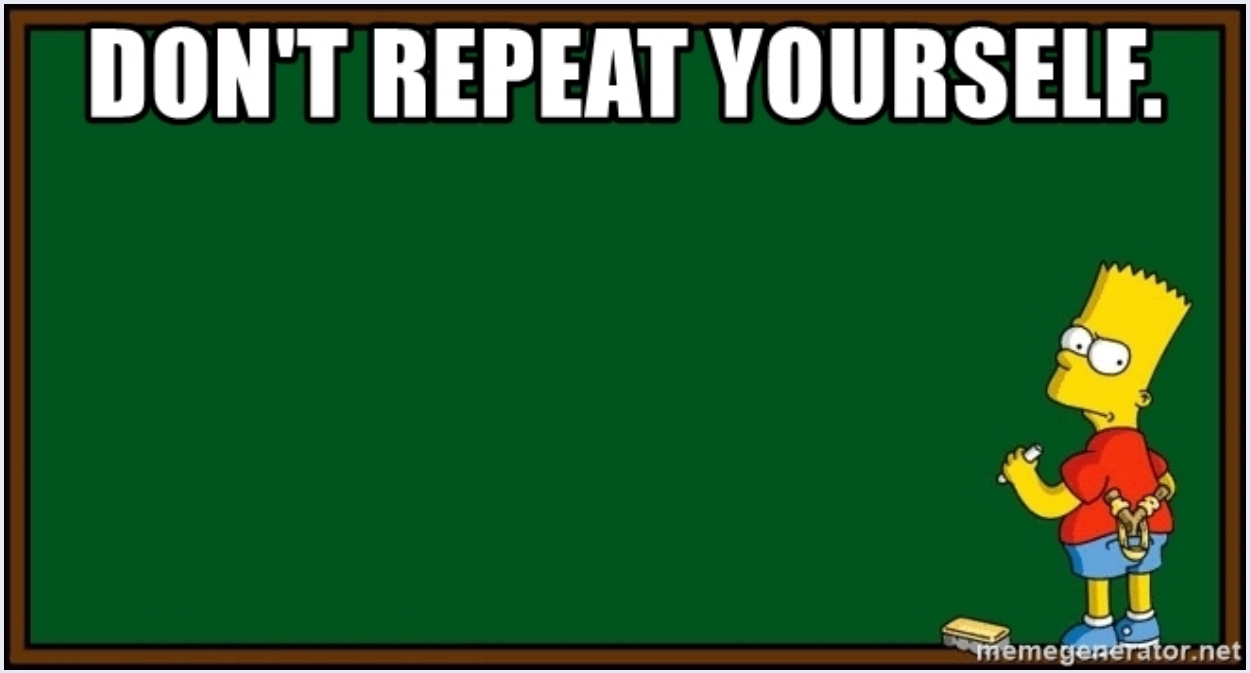


Based on this nice post

https://luminousmen.com/post/what-are-the-best-engineering-principles

Don’t Repeat Yourself (DRY)

If any code occurs more than twice in the codebase, you should think of moving it in a separate function. In fact, you should consider creating a separate method even if you encounter repetition a second time.



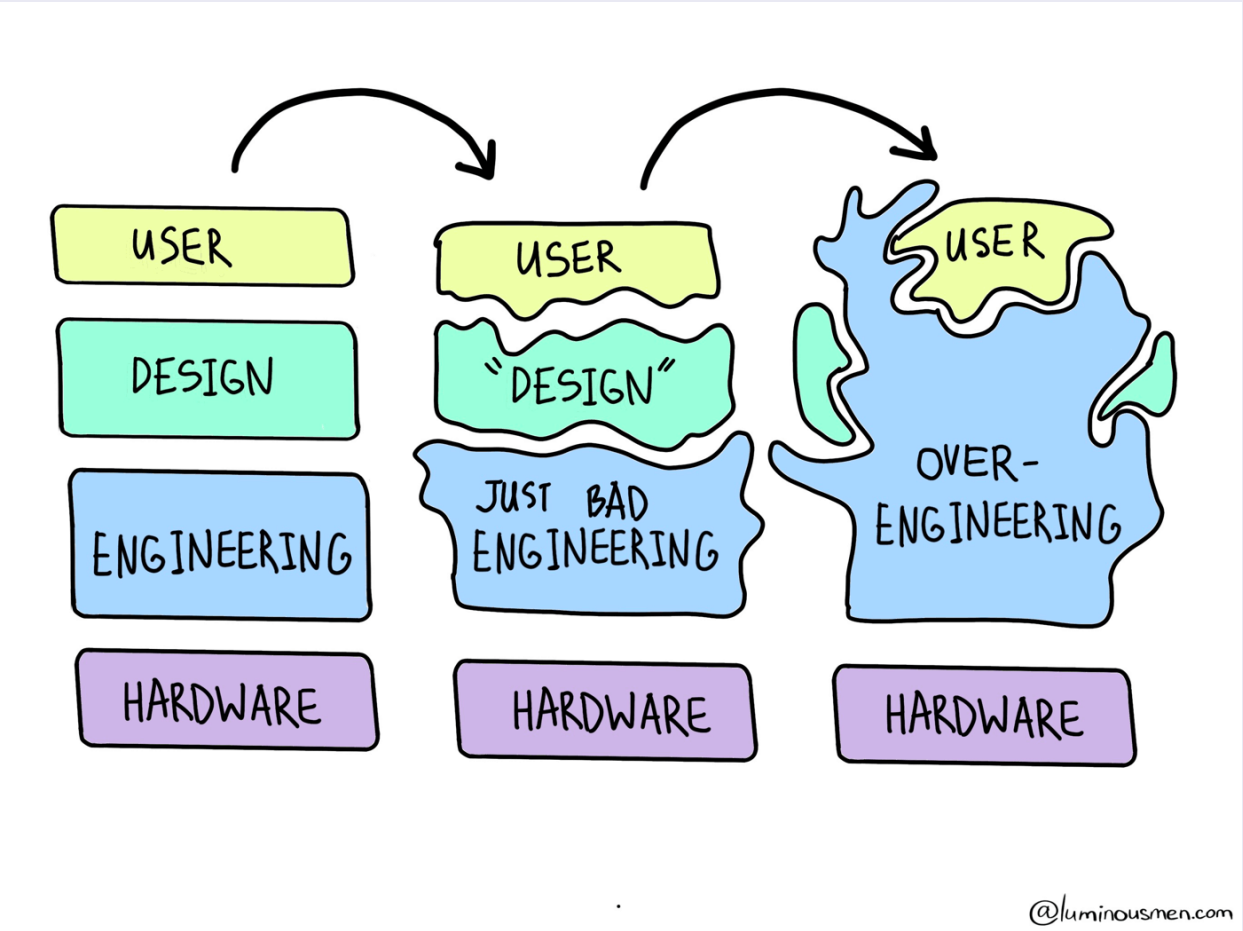
What Are The Best Software Engineering Principles?

Keep It Simple -Stupid- (KISS)

Some think that this idea transformed from Occam's Razor philosophical principle. You can interpret it as follows: one should not create extra entities to the system without a strong necessity. It is always a good idea to first consider the usefulness of adding another method/class/tool/process, etc.

You Aren't Gonna Need It (YAGNI)

Don't implement all the "necessary" (most likely unnecessary) functionality at once from the very beginning of the project.



What Are The Best Software Engineering Principles?

Avoid Premature Optimization

“Premature optimization is the root of all evil (or at least most of it) in programming” — Donald Knuth
Watch Knuth on a talk with Lex Friedman https://www.youtube.com/watch?v=EE1R8FYUJm0

Principle Of Least Astonishment

This principle means that your code should be intuitive and obvious, and not surprise another developer when reviewing the code.

Law of Demeter (Olympian goddess of the harvest and agriculture)

The basic idea here is to divide the areas of responsibility between classes and encapsulate the logic within a class, method, or structure.

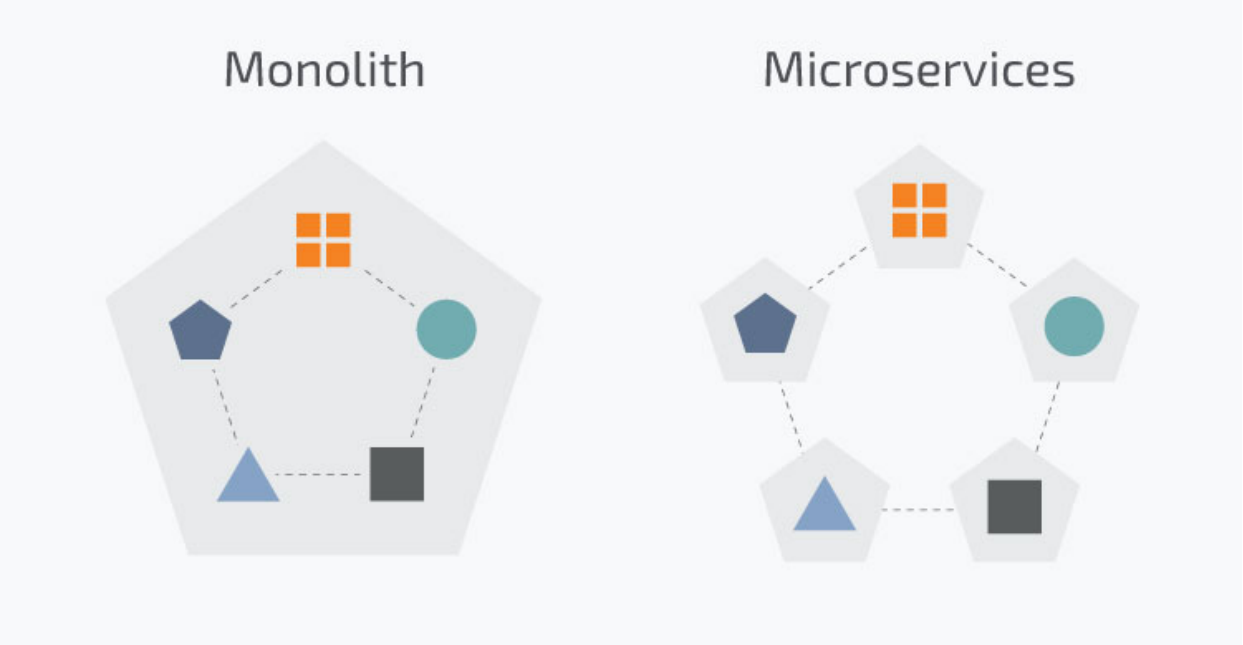
1. **Decoupling** You should try to reduce the number of connections between different classes or entities
2. **Cohesion** The associated classes must be in one module/package/directory

SOLID - create code that is easy to maintain and extend over time

- Single responsibility states that every module or class should have responsibility for a single part of the functionality and that responsibility should be entirely encapsulated by the class
- Open-closed states that software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification
- Liskov substitution states that any inherited class should complement (substitutable), not replace, the behavior of the base class
- Interface segregation states that no client of the class should be forced to depend on methods it does not use
- Dependency inversion says that programmers should work at the interface level and not at the implementation level

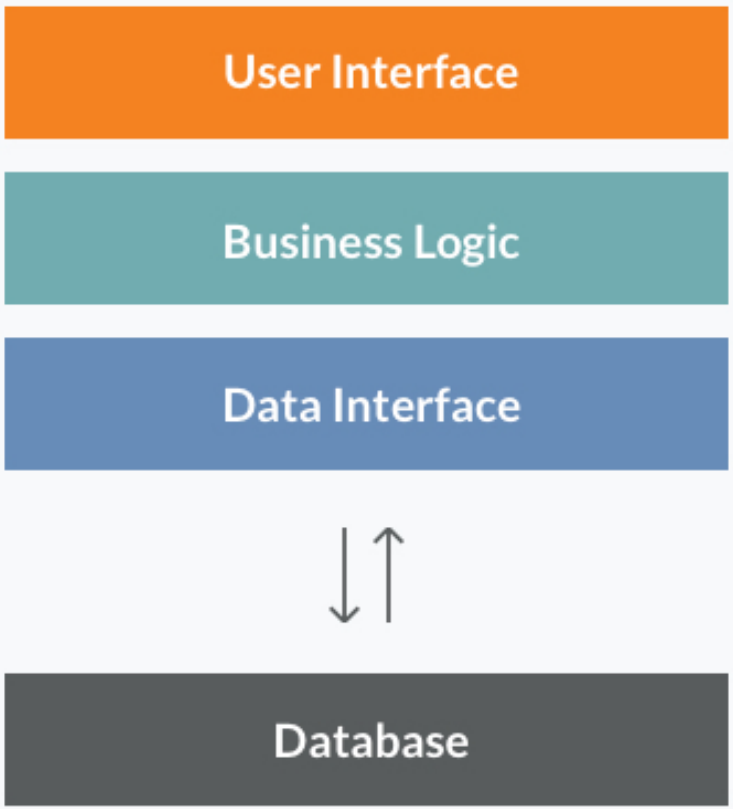
Monolithic vs. Microservices

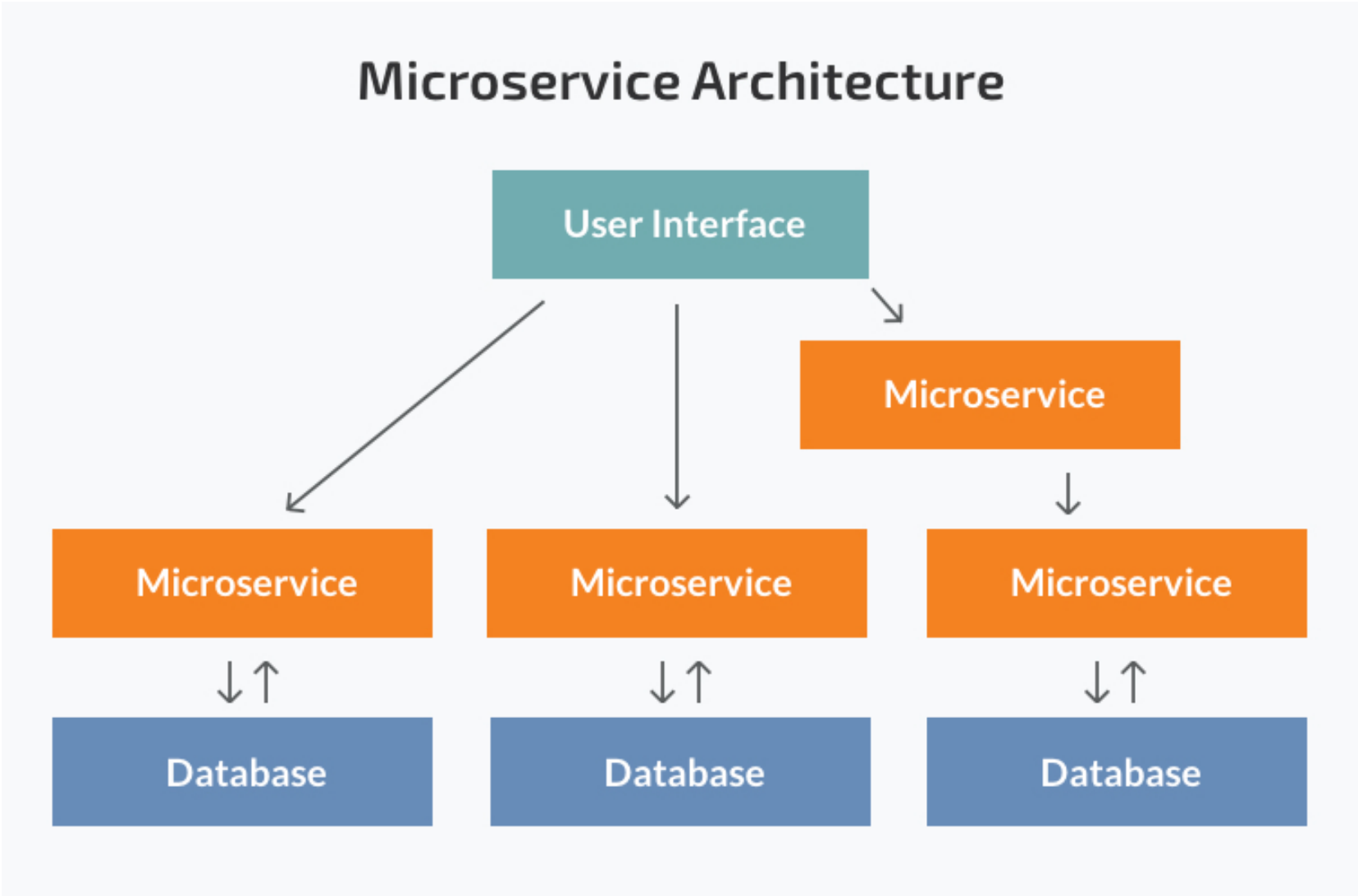
- Monolithic application is a single unified unit that contains all the logic in one entity
- Microservice architecture breaks the application down into a collection of smaller independent units



Further reading material

https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business


Monolithic Architecture
User Interface
Business Logic
Data Interface
Database


Microservice Architecture
User Interface
Microservice
Microservice
Microservice
Microservice
Database
Database
Database

- Dockerhub/Registry
- Dockerfile
- `docker build`
- `docker run`
- `docker ps`
- `docker network ls`
- `docker volumes`
- `docker expose ports`
- `docker images`
- `docker exec`
- `docker image prune -a`

Further training material

https://training.play-with-docker.com/alacart/
https://towardsdatascience.com/twenty-one-techniques-and-five-concepts-for-better-docker-usage-9ee135dccdc9

Advanced “Setting up a reverse proxy server” (How Docker makes our life easier)

“A very common scenario for developers, is to run their server behind a reverse proxy that sits in front of web servers and forwards client/frontend (e.g., web browser) requests to the web servers (”backend”). There are many reasons why you would want to do this but one of the main reasons is to run your API server on a different network or IP then your front-end application is on. You can then secure this network and only allow traffic from the reverse proxy server. For the sake of simplicity and space, I’ve created a simple frontend application in React.js and a simple backend API written in Node.js. Run the following command to pull the code from GitHub.”

Without docker this is not easy to do.

Soon we will see how Docker simplifies the process of building a server that run a reverse proxy

Go through (reverse proxy, react, nginx):

https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/

Using Docker nginx walkthrough an offical tutorial together

https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/

1. `docker run -it --rm -d -p 8080:80 --name web nginx`
2. `curl http://localhost:8080`
3. `docker stop web`
4. Add `index.html` to local site-content and map it to `/usr/share/nginx/html` (https://gist.github.com/chrisvfritz/bc010e6ed25b802da7eb)
5. `docker run -it --rm -d -p 8080:80 --name web -v ~/site-content:/usr/share/nginx/html nginx`
6. Doing stuff via Dockerfile (`docker build -t webserver`):

```
FROM nginx:latest
COPY ./index.html /usr/share/nginx/html/index.html
```

7. `docker run -it --rm -d -p 8080:80 --name web webserver`

Recap - linux commands everyone should know

host/network commands

- `ip`
- `ifconfig`
- `hostname`
- `whoami`
- `uname`
- `ping`

file-realted commands

- `mkdir` `rmdir` `cp` `mv`, `rm`
- `cd` `ls -l`
- `find`
- `wc`
- `xxd`
- `du -h /`
- `chown` `chmod`

archives:

- `zip`, `tar` `gzip`, `unzip`, `gunzip`
- `tar -czvf name-of-archive.tar.gz /path/to/dir-or-file`
  `c` - create, `v` verbose, `f` - allow to chose the name

Recap - linux commands everyone should know

more commands

- `cat` `touch` `echo`
- `locate` `whereis` `which` `find`
- `grep`
- `df` `du`
- `awk` `head` `tail`
- `diff`
- `jobs` (to see background jobs `command&`)
- `kill` (sending signals to processes)

`SIGTERM (15)` – requests the job to stop

`SIGKILL (9)` – forces programs to stop

- `wget` `curl`
- `top`, `htop`, `brew` (should be installed), `apt-install`

How microservices talk to each other

What is API?

API stands for Application Programming Interface. This interface allows users to build upon another application's functionality.

What is web API?

Web API is when other SW services uses other application's/service's functionbality over the web/network.

What is HTTP?

HTTP stands for Hypertext Transfer Protocol: an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems. - http://facebook.com - https://facebook.com

How microservices talk to each other

GET method

```
GET /microservice/v1/function?param1=value1&param=value2
```

POST method

```
POST /microservice/v1/function HTTP/1.1
Host: localhost

param1=value1&param=value2
```

How do we perform HTTP requests (postman and cli)

1. Postman https://web.postman.co/home
2. `curl` or `wget` in the command line

How do we perform HTTP requests (python)

3. `requests` library in python:

```
>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>> r.status_code
200
>> r.headers['content-type']
'application/json; charset=utf8'
>> r.encoding
'utf-8'
>> r.text
'{"type":"User"...'
>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

4. `httpx` library in python https://www.python-httpx.org/quickstart/

Demo performing HTTP reqeusts `httpx` vs. `curl`

1. Perform GET https://httpbin.org/get
2. Perform POST `https://httpbin.org/post` with `data={'key': 'value'}`
   - Note that POST/GET could be “overloaded” (have the same endpoint)

How microservices talk to each other

What is a REST API?

“When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it’s language-agnostic, as well as readable by both humans and machines”

Read more about REST and HTTP

https://www.redhat.com/en/topics/api/what-is-a-rest-api
https://www.educative.io/blog/what-are-rest-apis

In class hands-on session (training for Ex1)

Please complete due next class (March 7th, 2022) and use Discord for help

1. Create a remote git repo on our organization GitHub https://github.com/EASS-HIT-2022/ (private/public)
2. Name the repo `http-api-demo-<your github name>`
3. Include a README, Dockerfile, client.py files
4. In client.py include at least two POST/GET requests from httpbin demo HTTP API (http://httpbin.org/):
   - POST to any endpoint of your choice (e.g., http://httpbin.org/post)
   - GET to any endpoint of your choice (e.g., http://httpbin.org/get)
5. Make a Dockerfile that execute client.py on startup and prints the status and output from the http requests it performs from step 3. Helpful snippet:

```
FROM ubuntu
RUN  apt-get update
RUN  apt-get -y install python
CMD ["echo", "Hello, EASS 2022"]
```

6. Now, create a second Dockerfile in your git repo `localhost.Dockerfile` which call to the local hosted httpbin and call it via http://localhost: `docker run -p 80:80 kennethreitz/httpbin`
7. build the second docker image. Useful command:

```
docker build -t tab ./ -f localhost.Dockerfile
```

Hello World, FastAPI

Install

1. `pip install fastapi`
2. `pip install "uvicorn[standard]"` Uvicorn is an ASGI (Asynchronous Server Gateway Interface) web server implementation for Python

code of the server (`main.py`)

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def get_root():
    return {"message": "Hello World", "method": "GET"}

@app.post("/")
async def post_root():
    return {"message": "Hello World", "method": "POST"}
```

Hello World, FastAPI

Running the server

```
uvicorn main:app --reload
```

read more

https://fastapi.tiangolo.com/tutorial/

Ex 1 (Due April 10, 2022)

1. Build only the backend (using FastAPI)
2. Include a Dockerfile, README and the source code of the app
3. Be OOP-friendly (recall the SOLID principle) and use `pydantic`
4. Include both integartion and unit tests inside using `pytest`, `httpx`, or `pip install docker` (you may use some bash scripting as well). The idea is to be robust, simple and test the whole system wisely and efficently.

Suggested layout of the repo:

```
.
|- app
|              |- main.py
|              |- unit_tests.py
|              |- requirements.txt
|
|- integration_test.py
|- Dockerfile
|- README.md
```

List of ideas ideas for projects

- Building the backend of a voting app
- Personal wallet (keep expenses, images)
- Weather application
- The backend of a US/IL stock viewer analyzer webapp
- The backend of a twitter summarizer webapp (focus on one field e.g., stock symbols)

Good luck to all of us

Be active on EASS discord and try to learn and help each other as much as you can.

Triage

https://docs.microsoft.com/en-us/visualstudio/docker/tutorials/docker-tutorial
https://github.com/docker/awesome-compose/tree/master/fastapi
https://luminousmen.com/post/what-are-the-best-engineering-principles
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Concepts